



Programación Python para Big Data

**Lección 2: Creación de contenedores
(Dockers, Docker-compose, Kubernetes)**

ÍNDICE

Lección 2: Creación de contenedores (Dockers, Docker-compose, Kubernetes).....	2
Presentación y objetivos.....	2
1. Información sobre Docker	3
1.1 ¿Cómo funciona Docker?.....	3
1.2 Ventajas de los contenedores Docker.....	3
1.3 ¿Hay limitaciones para el uso de Docker?	5
1.4 ¿Los contenedores Docker son realmente seguros?	5
2. Crear una imagen Docker	6
2.1 Eliminación de imágenes.....	12
2.2 Imágenes de repositorio Docker hub.....	13
3. Docker-compose.....	17
4. Introducción a Kubernetes	21
5. Puntos claves	25

Lección 2: Creación de contenedores (Dockers, Docker-compose, Kubernetes)

PRESENTACIÓN Y OBJETIVOS

En esta lección aprenderemos a realizar un contenedor para ejecutar FastAPI mediante el uso de Docker y docker-compose. Y una breve introducción a Kubernetes.



Objetivos

- | Saber qué es un docker y cómo crear una imagen, ejecutarla y su gestión
- | Saber qué es docker-compose y cómo crear una imagen.
- | Conocer qué es Kubernetes.

1. INFORMACIÓN SOBRE DOCKER

Docker es crear contenedores ligeros y portables para las aplicaciones software que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues.

Obtiene flexibilidad con estos contenedores: puede crearlos, implementarlos, copiarlos y moverlos de un entorno a otro, lo cual le permite optimizar sus aplicaciones para la nube.

1.1 ¿Cómo funciona Docker?

El propósito de los contenedores es la capacidad de ejecutar varios procesos y aplicaciones por separado, además de conservar la seguridad que tendría como sistemas separados.

Docker, ofrecen un modelo de implementación basado en imágenes. Esto permite compartir una aplicación, o un conjunto de servicios, con todas sus dependencias en varios entornos. Docker también automatiza la implementación de la aplicación.

Docker es fácil de usar y único, permite la capacidad de implementar rápidamente y control sobre las versiones y su distribución.

1.2 Ventajas de los contenedores Docker

Modularidad

El enfoque Docker para la creación de contenedores se centra en la capacidad de tomar una parte de una aplicación, para actualizarla o repararla, sin necesidad de tomar la aplicación completa. Está basado en microservicios, puede compartir procesos entre varias aplicaciones de la misma forma que funciona la Arquitectura Orientada al Servicio (SOA).

Control de versiones de imágenes y capas

Cada archivo de imagen de Docker se compone de una serie de capas. Estas capas se combinan en una sola imagen. Una capa se crea cuando la imagen cambia. Cada vez que un usuario especifica un comando, como ejecutar o copiar, se crea una nueva capa.

Docker reutiliza estas capas para construir nuevos contenedores, lo cual hace mucho más rápido el proceso de construcción. Los cambios intermedios se comparten entre imágenes, mejorando aún más la velocidad, el tamaño y la eficiencia.

El control de versiones es inherente a la creación de capas. Cada vez que se produce un cambio nuevo, básicamente, tiene un registro de cambios incorporado: control completo de sus imágenes de contenedor.

Restauración

La imagen tiene capas pudiendo restaurarla a la versión anterior. Esto es compatible con un enfoque de desarrollo ágil y permite hacer realidad la integración e implementación continuas (CI/CD) desde una perspectiva de las herramientas.

Implementación rápida

Los contenedores basados en Docker pueden reducir el tiempo de implementación a segundos. Al crear un contenedor para cada proceso, puede compartir rápidamente los procesos similares con nuevas aplicaciones.

Además, con la velocidad de implementación, puede crear y destruir la información creada por sus contenedores sin preocupación, de forma fácil y rentable.

Por lo tanto, la tecnología Docker es un enfoque más granular y controlable, basado en microservicios, que prioriza la eficiencia.

1.3 ¿Hay limitaciones para el uso de Docker?

En sí mismo, Docker es una excelente herramienta para la gestión de contenedores individuales. Al comenzar a utilizar cada vez más contenedores y aplicaciones en contenedores, divididas en cientos de piezas, la gestión y la organización se pueden tornar muy difíciles. Finalmente, debe retroceder y agrupar los contenedores para ofrecer servicios, como redes, seguridad, telemetría, etc., en todos sus contenedores. Es aquí donde aparece Kubernetes.

1.4 ¿Los contenedores Docker son realmente seguros?

El daemon de Docker también puede ser una preocupación en materia de seguridad. Para usar y ejecutar los contenedores Docker, es muy probable que utilice el daemon de Docker, un tiempo de ejecución persistente para los contenedores. El daemon de Docker requiere privilegios de raíz, por lo que se debe prestar especial atención a quiénes obtienen acceso al proceso y en dónde reside este. Por ejemplo, un daemon local tiene una superficie de ataque más pequeña que uno que se encuentra en un sitio más público, como un servidor web.

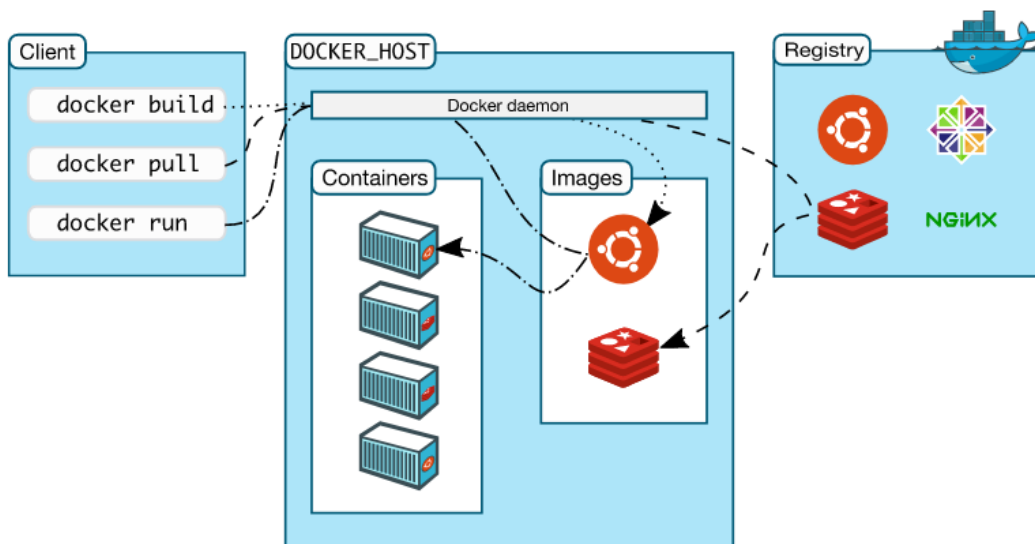


Figura 1.1. Estructura y descripción de Docker

2. CREAR UNA IMAGEN DOCKER

Lo primero será necesario crear una imagen, para ello realizaremos un script de Python donde crearemos una aplicación de FastAPI.

Para ello creamos una carpeta con el nombre app dentro de la carpeta (docker) donde vamos a crear nuestra imagen:

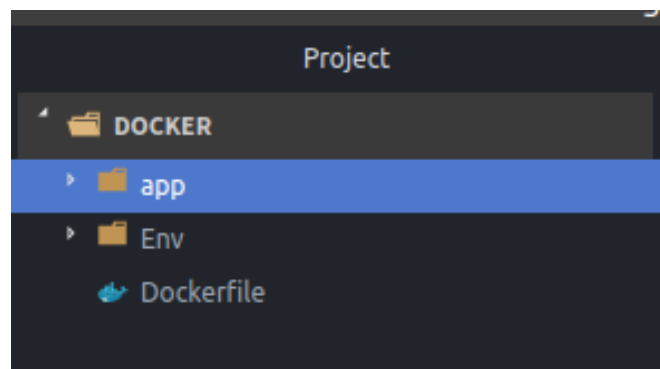


Figura 2.1 Imagen de la estructura que vamos a crear para nuestro primer docker

Dentro de esa carpeta creamos el archivo main.py donde crearemos una aplicación de FastAPI.

```
from fastapi import FastAPI

app = FastAPI()

@app.get('/')
async def root():
    return {'message': "Bienvenido a FastAPI"}
```

```

main.py
1  from fastapi import FastAPI
2
3  app = FastAPI()
4
5
6  @app.get('/')
7  async def root():
8      return {'message': "Bienvenido a FastAPI"}
9

```

Figura 2.2 Archivo main.py de nuestra aplicación FastAPI

Instalamos en nuestro entorno virtual:

```

pip install uvicorn[standard]

pip install fastapi

```

Ejecutamos la aplicación para ver que funciona:

```

uvicorn main:app --reload

```

```

(Evn) isabel@isabel-SVE1512E1EW:~/atom/proyectos_finales/FastApi$ uvicorn main:app --reload
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [11311] using watchgod
INFO:     Started server process [11313]
INFO:     Waiting for application startup.
INFO:     Application startup complete.

```

Figura 2.3 Ejecutamos la aplicación FastAPI

Creamos un archivo con nombre Dockerfile, este es un documento de texto que contiene todos los comandos que un usuario podría llamar en línea para la imagen. Se habla de construir nuestra propia imagen Docker cuando ejecutamos el comando docker build. Docker lee estas instrucciones y los ejecuta consecutivamente y crea una imagen de docker como resultado.

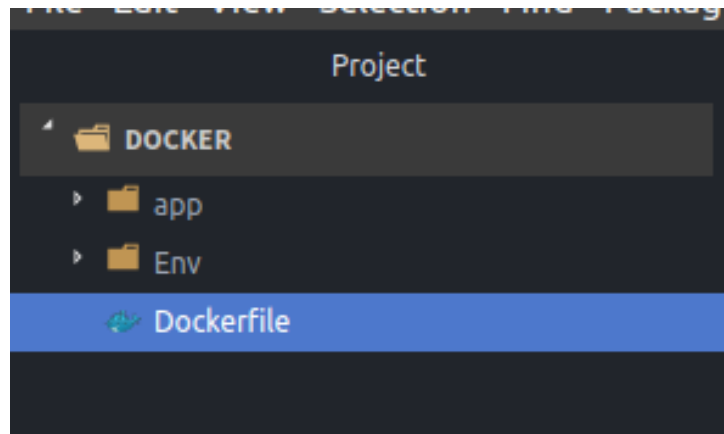


Figura 2.4 Estructura de nuestro ejemplo Docker

En el archivo Dockerfile escribimos:

```
# Versión de nuestro dockerfile a 1
# syntax=docker/dockerfile:1

# Necesitamos 3.8 de python mediante una imagen.
FROM python:3.8

# Instalamos las librerías necesarias
RUN pip install fastapi uvicorn

# puerto que vamos a poner
EXPOSE 80

# La imagen está basada en Python 3.8 y tiene instaladas las dependencias
# Añadimos el código fuente a la imagen
COPY ./docker /app

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "80"]
```

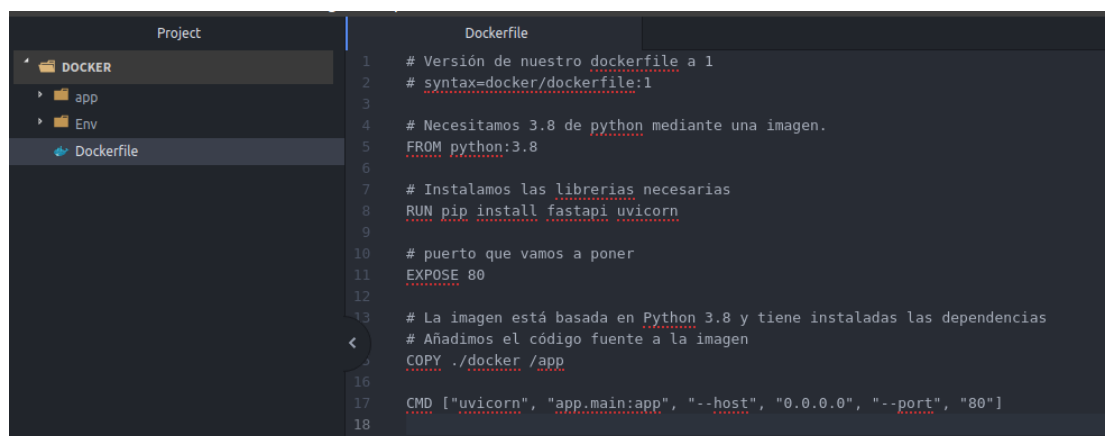


Figura 2.5 Ejemplo de Dockerfile para FastAPI

Construimos la imagen llamamos eip/fastapi, esta ejecutará el archivo Dockerfile creando la estructura necesaria para su ejecución:

```
sudo docker build -t eip/fastapi . /
```

```
isabel@isabel-SVE1512E1EW:~/atom/docker$ sudo docker build -t eip/fastapi . /
Sending build context to Docker daemon 132.3MB
Step 1/5 : FROM python:3.8
----> 95a6101eab8f
Step 2/5 : RUN pip install fastapi uvicorn
----> Running in af664288f326
Collecting fastapi
  Downloading fastapi-0.65.2-py3-none-any.whl (51 kB)
Collecting uvicorn
  Downloading uvicorn-0.14.0-py3-none-any.whl (50 kB)
Collecting starlette==0.14.2
  Downloading starlette-0.14.2-py3-none-any.whl (60 kB)
Collecting pydantic<1.7.1,!=1.7.1,!=1.7.2,!=1.7.3,!=1.8,!=1.8.1,<2.0.0,>=1.6.2
  Downloading pydantic-1.8.2-cp38-cp38-manylinux2014_x86_64.whl (13.7 MB)
Collecting typing-extensions>=3.7.4.3
  Downloading typing_extensions-3.10.0.0-py3-none-any.whl (26 kB)
Collecting h11>=0.8
  Downloading h11-0.12.0-py3-none-any.whl (54 kB)
Collecting click>=7.*
  Downloading click-8.0.1-py3-none-any.whl (97 kB)
Collecting asgiref>=3.3.4
  Downloading asgiref-3.3.4-py3-none-any.whl (22 kB)
Installing collected packages: typing-extensions, starlette, pydantic, h11, click, asgiref, uvicorn, fastapi
Successfully installed asgiref-3.3.4 click-8.0.1 fastapi-0.65.2 h11-0.12.0 pydantic-1.8.2 starlette-0.14.2 typing-extensions-3.10.0.0 uvicorn-0.14.0
WARNING: Running pip as root will break packages and permissions. You should install packages reliably by using venv: https://pip.pypa.io/warnings/venv
```

```
Removing intermediate container af664288f326
----> 355ecc028a7e
Step 3/5 : EXPOSE 80
----> Running in f25321d00119
Removing intermediate container f25321d00119
----> 9bcf3a69f6b4
Step 4/5 : COPY ./app /app
----> f096dd833396
Step 5/5 : CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "80"]
----> Running in 7c9729b1d567
Removing intermediate container 7c9729b1d567
----> 4fec14ee317e
Successfully built 4fec14ee317e
Successfully tagged eip/fastapi:latest
```

Figura 2.6 Construcción de nuestra imagen

Comprobamos que se ha creado la imagen:

```
sudo docker images
```

```
isabel@isabel-SVE1512E1EW:~/atom/docker$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
eip/fastapi    latest    4fec14ee317e   8 seconds ago  1GB
python         3.8       95a6101eab8f   2 hours ago    883MB
```

Figura 2.7 Imágenes creadas de Python y nuestra imagen eip/fastapi

Ejecutamos la imagen de docker (eip/fastapi):

```
sudo docker run -d --name fastapi -p 80:80 eip/fastapi
```

```
lsabel@lsabel-SVE1512E1EW:~/atom/docker$ sudo docker run -d --name fastapi -p 80:80 eip/fastapi
88b1ae17e9e4e0c34bbbed0d4680b1eaecaecf621d221d2be15963721e6c9854f
```

Figura 2.8 Ejecutamos la imagen en segundo plano

Donde:

- | run para ejecutar el docker
- | -d para ejecutarlo en segundo plano.
- | --name nombre del docker
- | -p 80:80 puerto en el cual se ejecuta el docker interno y externo
- | eip/fastapi imagen que creamos anteriormente que sea llamada.

Comprobamos que el docker funciona:

```
sudo docker ps
```

```
lsabel@lsabel-SVE1512E1EW:~/atom/docker$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
88b1ae17e9e4   eip/fastapi "uvicorn app.main:ap..." 9 seconds ago  Up 9 seconds  0.0.0.0:80->80/tcp, :::80->80/tcp  fastapi
```

Figura 2.9 Comprobación de ejecución de la imagen

Comprobamos en el navegador que funciona: <http://localhost:80>

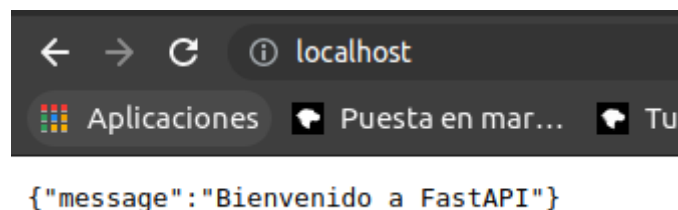


Figura 2.10 Comprobación de ejecución de la imagen en el navegador

http://localhost:80/docs

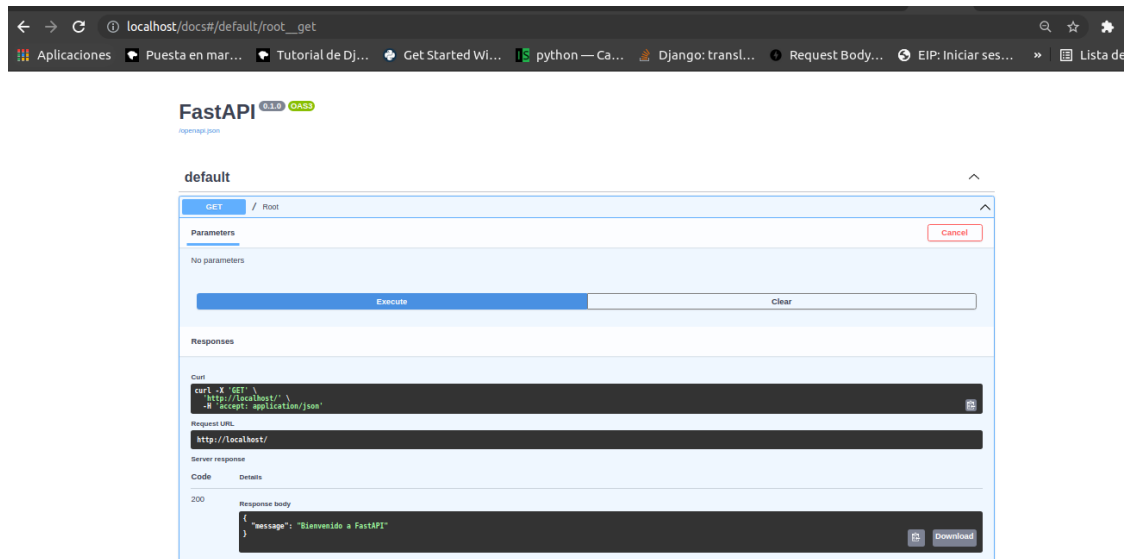


Figura 2.11 Comprobación de ejecución de la imagen en el navegador

Para parar el docker:

```
sudo docker stop fastapi
```

ó

```
Sudo docker kill fastapi
```

```
isabel@isabel-SVE1512E1EW:~/atom/docker$ sudo docker stop fastapi
fastapi
```

Figura 2.12 Para el proceso de ejecución de la imagen

Comprobamos que ya no está ejecutado:

```
sudo docker ps
```

```
isabel@isabel-SVE1512E1EW:~/atom/docker$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
```

Figura 2.13 Imagen de procesos activos

2.1 Eliminación de imágenes

Comprobamos el sistema docker formado por imágenes, contenedores creados, *volumes* (networks) y caché:

```
sudo docker system df
```

```
isabel@isabel-SVE1512E1EW:~/atom/docker$ sudo docker system df
TYPE                TOTAL        ACTIVE        SIZE        RECLAIMABLE
Images              2            1            1GB        883.4MB (88%)
Containers          1            0            96.86kB    96.86kB (100%)
Local Volumes       0            0            0B         0B
Build Cache         0            0            0B         0B
```

Figura 2.14 Información sobre el sistema Docker

Eliminar los containers:

```
sudo docker system prune
```

```
isabel@isabel-SVE1512E1EW:~/atom/docker$ sudo docker system prune
WARNING! This will remove:
- all stopped containers
- all networks not used by at least one container
- all dangling images
- all dangling build cache

Are you sure you want to continue? [y/N] y
Deleted Containers:
88b1ae17e9e4e0c34bbcd0d4680b1eaecaecf621d221d2be15963721e6c9854f

Total reclaimed space: 96.86kB
```

Figura 2.15 Eliminación de los containers.

Elimino la imagen:

```
sudo docker image rm eip/fastapi
```

```
isabel@isabel-SVE1512E1EW:~/atom/docker$ sudo docker image rm eip/fastapi
Untagged: eip/fastapi:latest
Deleted: sha256:4fec14ee317eaa23e363bb023ad687497f9a574ecc3081480cbb868ea8e99417
Deleted: sha256:f096dd83339677e73cd8b3cb3eb4b827b94d77a35568d1d6dd16128451ea2d41
Deleted: sha256:b07cfc86a1369ed088b1a00301e118e705d4080514f93558e52aa739897c38d7
Deleted: sha256:9bcf3a69f6b43463a0e8086ab19ecaaa0baf81be54b9a943429d7e266fa86bb3
Deleted: sha256:355ecc028a7e72dd6ea09154c0aac93544450bf3a5f9223eac2beaaa39303acc
Deleted: sha256:b229a5d32dae750331d9305abb9aa356e84578ae96a9a47f5f58d9280f16cc20
```

Figura 2.16 Eliminación de las imágenes.

Comprobación de las imágenes que nos quedan:

```
isabel@isabel-SVE1512E1EW:~/atom/docker$ sudo docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
python        3.8       95a6101eab8f   2 hours ago   883MB
```

Figura 2.17 Información de las imágenes disponibles

Sólo nos queda la imagen de Python que puede ser útil para otros desarrollos.

2.2 Imágenes de repositorio Docker hub

Vamos a la página oficial de docker hub:

<https://hub.docker.com/search?q=fastapi&type=image>

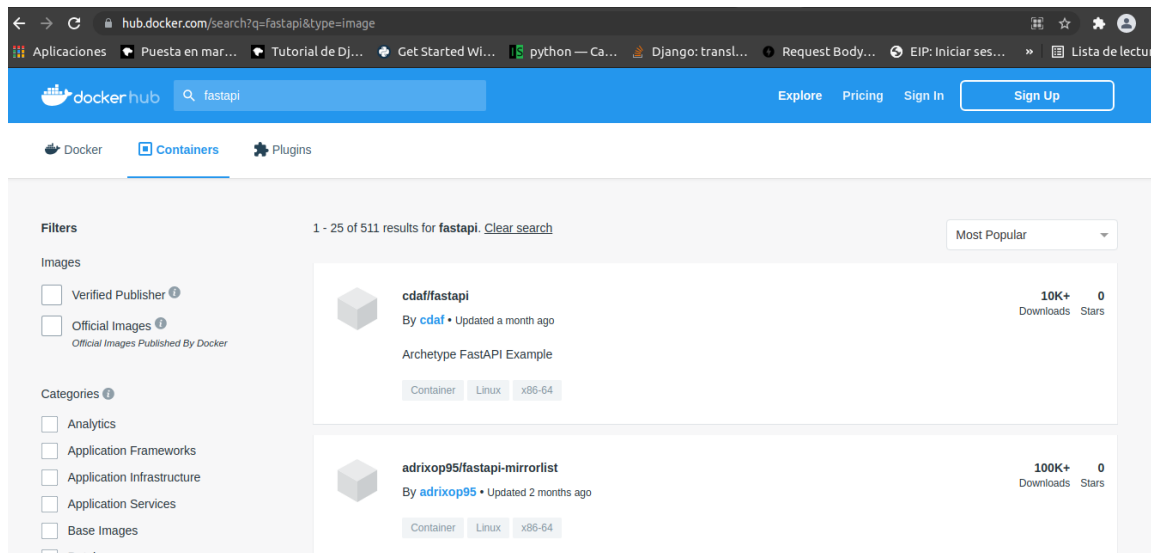


Figura 2.18 Búsqueda en docker hub de repositorio de FastAPI

Escogemos la segunda opción:

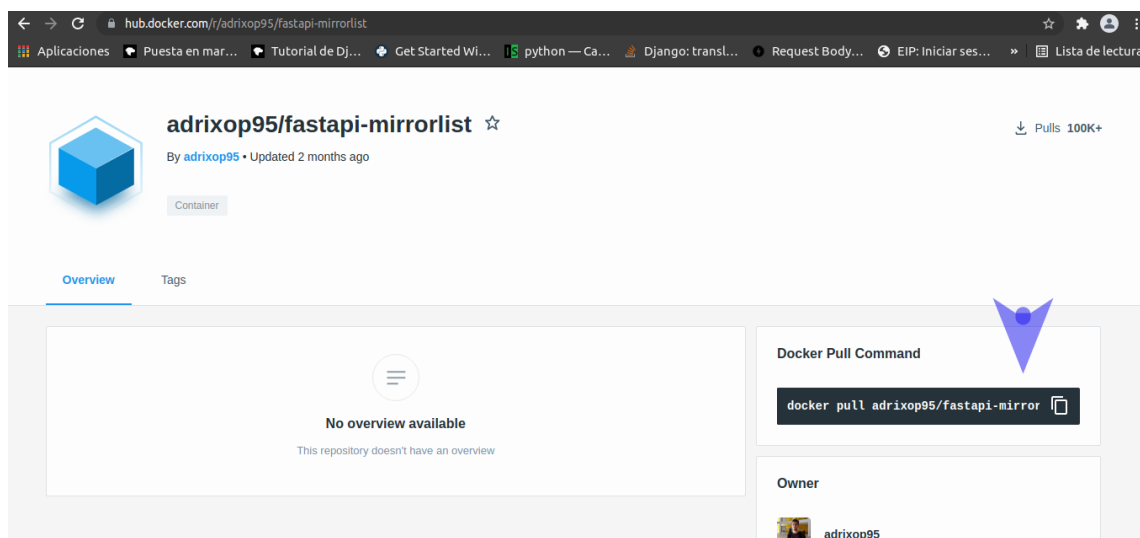


Figura 2.19 Búsqueda en docker hub de repositorio de FastAPI

Copiamos la instrucción y ejecutamos para que nos descargue/construya la imagen:

```
sudo docker pull adrixop95/fastapi-mirrorlist
```

```
isabel@isabel-SVE1512E1EW:~$ sudo docker pull adrixop95/fastapi-mirrorlist
[sudo] contraseña para isabel:
Using default tag: latest
latest: Pulling from adrixop95/fastapi-mirrorlist
540db60ca938: Pull complete
d037ddac5dde: Pull complete
05d0edf52df4: Pull complete
54d94e388fb8: Pull complete
b25964b87dc1: Pull complete
90c3176c6f41: Pull complete
6178cd98500e: Pull complete
Digest: sha256:c20c932aba27983d4cb9fcf0f494b2da3b42d4467faa3d97a498ddd4789d5f2d
Status: Downloaded newer image for adrixop95/fastapi-mirrorlist:latest
docker.io/adrixop95/fastapi-mirrorlist:latest
```

Figura 2.20 Descarga y construcción de la imagen

Se crea la imagen:

```
Sudo docker images
```

```
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS        NAMES
isabel@isabel-SVE1512E1EW:~$ sudo docker images
REPOSITORY          TAG          IMAGE ID          CREATED        SIZE
adrixop95/fastapi-mirrorlist  latest      50b4cd4bd3c2     2 months ago  77.1MB
```

Figura 2.21 Imagen creada

Ejecutamos con:

```
sudo docker run --name fastapi -p 8000:8000 adrixop95/fastapi-mirrorlist
```

```
isabel@isabel-SVE1S12E1EW:~$ sudo docker run --name fastapi -p 8000:8000 adrixop95/fastapi-mirrorlist
[2021-06-30 17:07:44 +0000] [1] [INFO] Starting gunicorn 20.1.0
[2021-06-30 17:07:44 +0000] [1] [INFO] Listening at: http://0.0.0.0:8000 (1)
[2021-06-30 17:07:44 +0000] [1] [INFO] Using worker: uvicorn.workers.UvicornWorker
[2021-06-30 17:07:44 +0000] [8] [INFO] Booting worker with pid: 8
[2021-06-30 17:07:44 +0000] [9] [INFO] Booting worker with pid: 9
[2021-06-30 17:07:44 +0000] [10] [INFO] Booting worker with pid: 10
[2021-06-30 17:07:44 +0000] [11] [INFO] Booting worker with pid: 11
[2021-06-30 17:07:45 +0000] [8] [INFO] Started server process [8]
[2021-06-30 17:07:45 +0000] [8] [INFO] Waiting for application startup.
[2021-06-30 17:07:45 +0000] [8] [INFO] Application startup complete.
[2021-06-30 17:07:45 +0000] [9] [INFO] Started server process [9]
[2021-06-30 17:07:45 +0000] [9] [INFO] Waiting for application startup.
[2021-06-30 17:07:45 +0000] [9] [INFO] Application startup complete.
[2021-06-30 17:07:45 +0000] [10] [INFO] Started server process [10]
[2021-06-30 17:07:45 +0000] [10] [INFO] Waiting for application startup.
[2021-06-30 17:07:45 +0000] [10] [INFO] Application startup complete.
[2021-06-30 17:07:45 +0000] [11] [INFO] Started server process [11]
[2021-06-30 17:07:45 +0000] [11] [INFO] Waiting for application startup.
[2021-06-30 17:07:45 +0000] [11] [INFO] Application startup complete.
```

Figura 2.22 Ejecución de la imagen

<http://127.0.0.1:8000/docs>

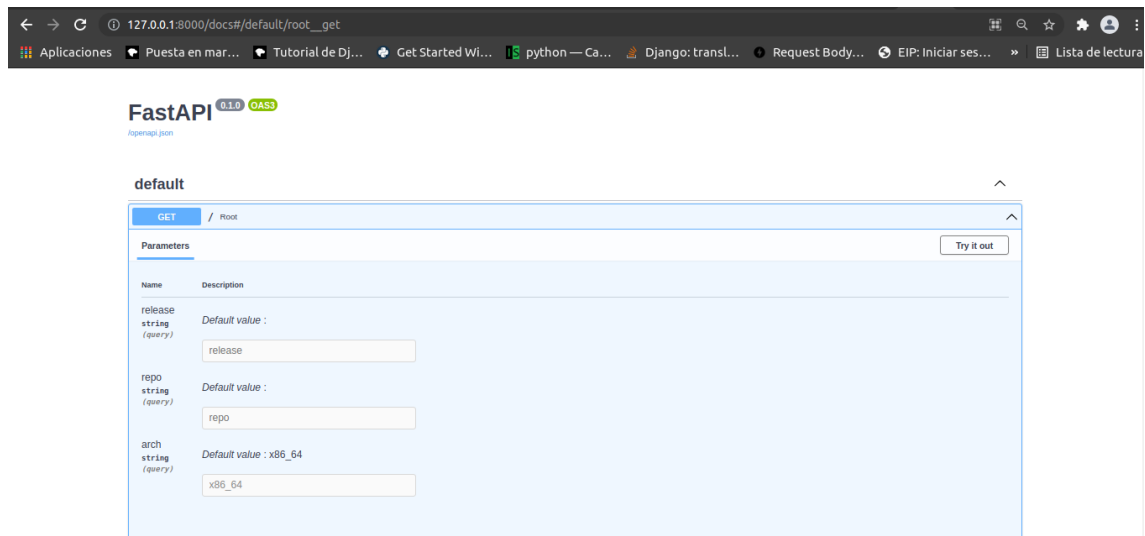


Figura 2.23 Ejecución de la imagen en el navegador

3. DOCKER-COMPOSE

Definición

Compose es una herramienta para definir y ejecutar multi-contenedores de aplicaciones Docker, con docker-compose se usa un archivo YAML para configurar la aplicación de los servicios, con un simple comando, tú puedes crear y ejecutar todos los servicios que hay en tu configuración (varias imágenes de Docker).

Ejecución

Moveremos nuestro "Dockerfile" dentro de una carpeta que hemos creado con nombre "project" y también la carpeta "app":

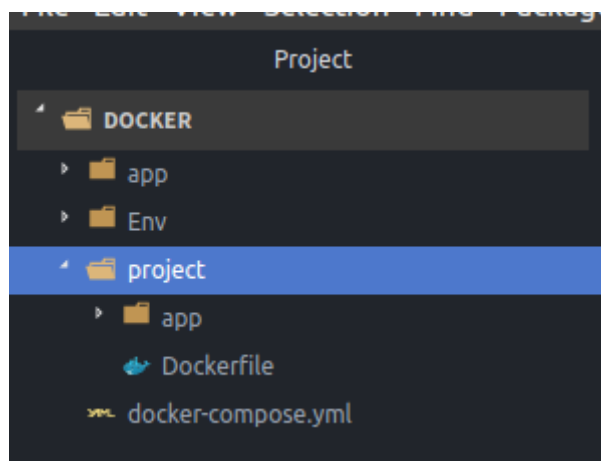


Figura 3.1 Estructura para ejecutar el docker-compose

Creamos nuestro archivo docker-compose.yml:

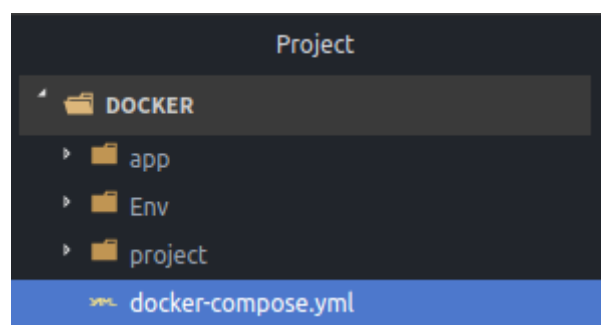


Figura 3.2 Crear el archivo docker-compose

Ponemos:

```
# versión de docker-compose que vamos a usar de formato usado
version: '3'

services:
  # nombre de nuestro servicio 'web'
  web:
    # donde tenemos nuestro archivo
    build: ./project
    # comando de ejecución
    command: uvicorn app.main:app --reload --workers 1 --host 0.0.0.0 --port 80
    # donde se va a guardar la información en el container
    volumes:
      - ./project:/usr/src/app
    # puertos activos externos e internos
    ports:
      - 80:80
    # entorno en este caso de desarrollo
    environment:
      - ENVIRONMENT=dev
      - TESTING=0
```

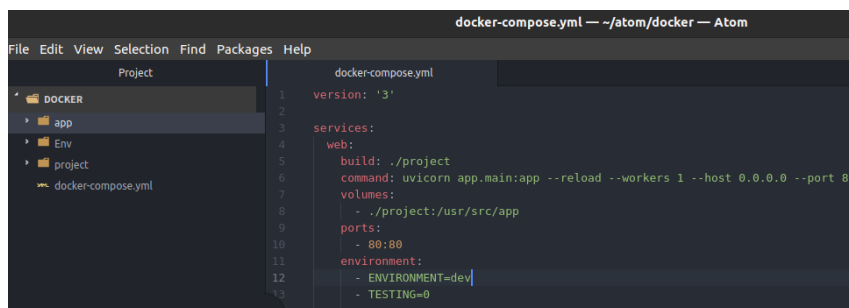


Figura 3.3 Archivo docker-compose

Creamos las imágenes:

```
sudo docker-compose build
```

```

lsabel@lsabel-SVE1512E1EW:~/atom/docker$ sudo docker-compose build
Building web
Step 1/5 : FROM python:3.8
3.8: Pulling from library/python
0bc3020d05f1: Pull complete
a110e5871660: Pull complete
83d3c0fa203a: Pull complete
a8fd09c11b02: Pull complete
14feb89c4a52: Pull complete
70752631d778: Pull complete
a528eeaeed26: Pull complete
01a2d9ea11b3: Pull complete
0f9055ba88f3: Pull complete
Digest: sha256:19e990af4dcaac714b4c6f816afa13d0678750ebc3ae3f7011fdc8715025817b
Status: Downloaded newer image for python:3.8
--> 95a6101eab8f
Step 2/5 : RUN pip install fastapi uvicorn
--> Running in 4f2c1086c24d
Collecting fastapi
  Downloading fastapi-0.65.2-py3-none-any.whl (51 kB)
Collecting uvicorn
  Downloading uvicorn-0.14.0-py3-none-any.whl (50 kB)
Collecting starlette==0.14.2
  Downloading starlette-0.14.2-py3-none-any.whl (60 kB)
Collecting pydantic!=1.7,!=1.7.1,!=1.7.2,!=1.7.3,!=1.8,!=1.8.1,<2.0.0,>=1.6.2
  Downloading pydantic-1.8.2-cp38-cp38-manylinux2014_x86_64.whl (13.7 MB)
Collecting typing-extensions>=3.7.4.3
  Downloading typing_extensions-3.10.0.0-py3-none-any.whl (26 kB)
Collecting h11>=0.8
  Downloading h11-0.12.0-py3-none-any.whl (54 kB)
Collecting click>=7.*
  Downloading click-8.0.1-py3-none-any.whl (97 kB)
Collecting asgiref<3.4
  Downloading asgiref-3.3.4-py3-none-any.whl (22 kB)
Installing collected packages: typing-extensions, starlette, pydantic, h11, click, asgiref, uvicorn, fastapi
Successfully installed asgiref-3.3.4 click-8.0.1 fastapi-0.65.2 h11-0.12.0 pydantic-1.8.2 starlette-0.14.2 typing-extensions-3.10.0.0 uvicorn-0.14.0
WARNING: Running pip as root will break packages and permissions. You should install packages reliably by using 'pip install --user'
Removing intermediate container 4f2c1086c24d
--> 2179b76d2abe
Step 3/5 : EXPOSE 80
--> Running in e88d23438d9
Removing intermediate container e88d23438d9
--> 7d8f63e9ee
Step 4/5 : COPY ./app /app
--> 3d39c6a80f46
Step 5/5 : CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "80"]
--> Running in e802ceba0408
Removing intermediate container e802ceba0408
--> ae245030c9f7
Successfully built ae245030c9f7
Successfully tagged docker_web:latest

```

Figura 3.4 Construir la imagen de docker

Comprobar que están las imágenes creadas:

```

lsabel@lsabel-SVE1512E1EW:~/atom/docker$ sudo docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
docker_web          latest          ae245030c9f7   5 minutes ago   1GB
python              3.8            95a6101eab8f   3 hours ago     883MB

```

Figura 3.5 Imágenes creadas

docker-compose up

```

lsabel@lsabel-SVE1512E1EW:~/atom/docker$ sudo docker-compose up
Creating network "docker_default" with the default driver
Creating docker_web_1 ... done
Attaching to docker_web_1
web_1 | INFO:      Uvicorn running on http://0.0.0.0:80 (Press CTRL+C to quit)
web_1 | INFO:      Started reloader process [1] using statreload
web_1 | INFO:      Started server process [8]
web_1 | INFO:      Waiting for application startup.
web_1 | INFO:      Application startup complete.
web_1 | INFO:      172.18.0.1:35822 - "GET /openapi.json HTTP/1.1" 200 OK
web_1 | INFO:      172.18.0.1:35820 - "GET / HTTP/1.1" 200 OK

```

Figura 3.6 Ejecución de la imagen

Comprobamos en el navegador: `http://localhost:80/docs`

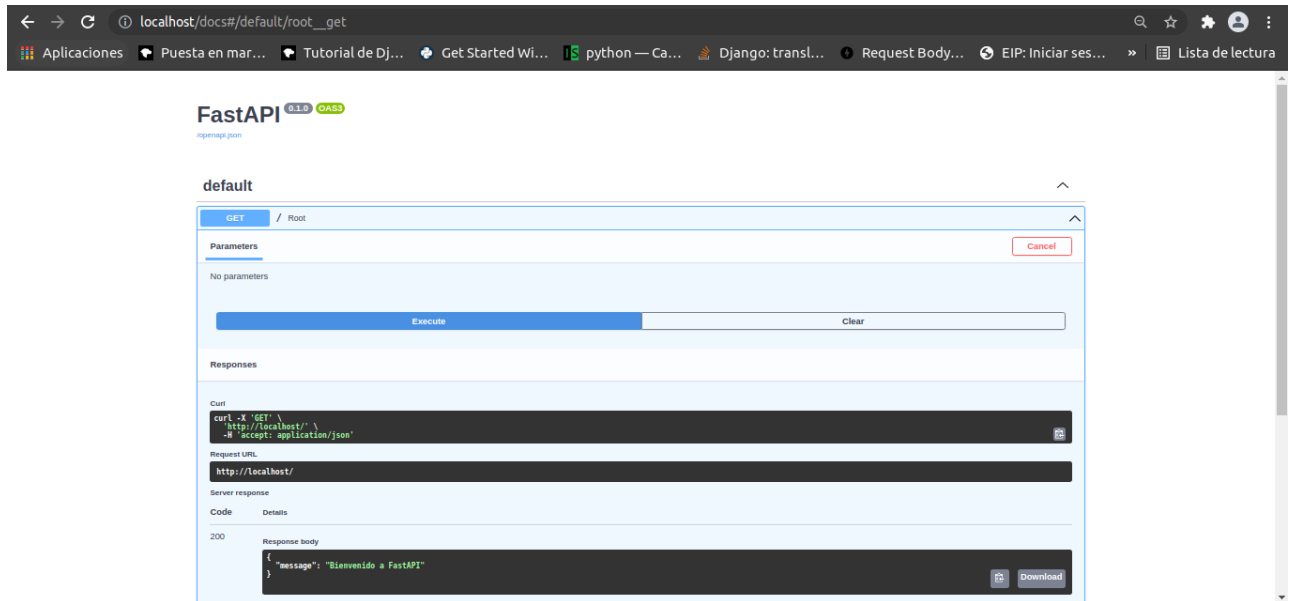


Figura 3.7 Ejecución de la aplicación

Para parar la aplicación:

`Ctrl + c`

```
isabel@isabel-SVE1512E1EW:~/atom/docker$ sudo docker-compose up
Creating network "docker_default" with the default driver
Creating docker_web_1 ... done
Attaching to docker_web_1
web_1 | INFO:      Uvicorn running on http://0.0.0.0:80 (Press CTRL+C to quit)
web_1 | INFO:      Started reloader process [1] using statreload
web_1 | INFO:      Started server process [8]
web_1 | INFO:      Waiting for application startup.
web_1 | INFO:      Application startup complete.
web_1 | INFO:      172.18.0.1:35822 - "GET /openapi.json HTTP/1.1" 200 OK
web_1 | INFO:      172.18.0.1:35820 - "GET / HTTP/1.1" 200 OK
^CGracefully stopping... (press Ctrl+C again to force)
Stopping docker_web_1 ... done
```

Figura 3.8 Parar la aplicación

Si queremos que quede en segundo plano ponemos

`docker-compose up -d`

Quedaría ejecutada igual que en el caso del docker.

4. INTRODUCCIÓN A KUBERNETES

Kubernetes es una plataforma open source para el automatizar el despliegue, escalado de las aplicaciones, así como las operaciones con los contenedores de aplicaciones, desarrollado por Google. Administra aplicaciones en contenedores en varios tipos de entornos físicos, virtuales y en la nube.

Kubernetes es una herramienta de contenedor altamente flexible para entregar de manera consistente aplicaciones complejas que se ejecutan en clústeres de cientos a miles de servidores individuales.

Es necesario conocer una serie de conceptos:

- | **Cluster:** Conjunto de máquinas físicas o virtuales y otros recursos utilizados por kubernetes.
- | **Nodo:** Una máquina física o virtual ejecutándose en kubernetes donde los *pods* pueden ser programados.
- | **Pod:** Son la unidad más pequeña desplegable que puede ser creada, programada y manejada por kubernetes.
- | **Replication Controller:** Se asegura de que el número especificado de réplicas del pod estén ejecutándose. Permite escalar de forma fácil los sistemas y maneja la re-creación de un pod cuando ocurre un fallo.
- | **Service:** Es una abstracción que define un conjunto de pods y la lógica para acceder a los mismos.

¿Por qué usar contenedores?

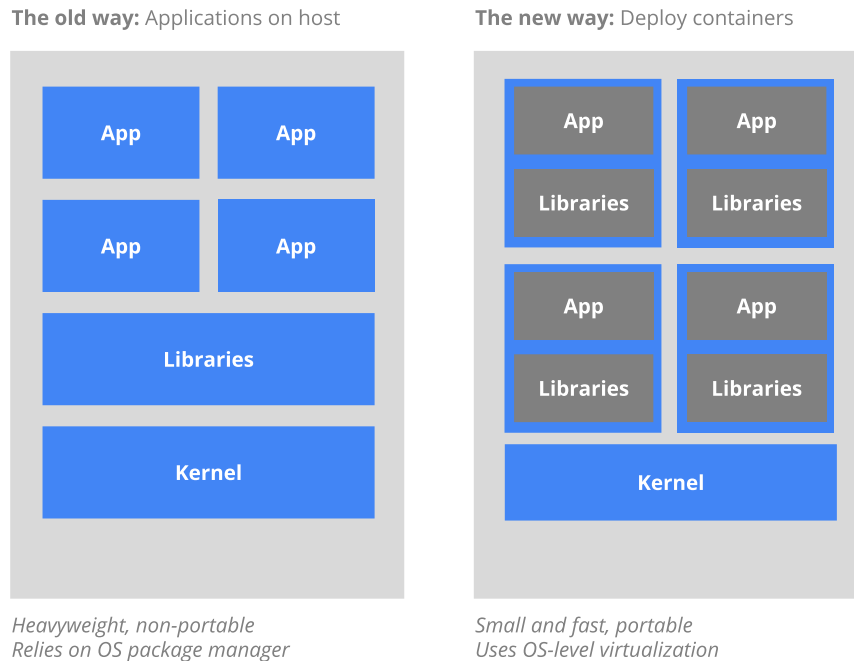


Figura 4.1 Diferencia entre usar Kubernetes o no

La Manera Antigua de desplegar aplicaciones era instalarlas en un servidor usando el administrador de paquetes del sistema operativo. La desventaja era que los ejecutables, la configuración, las librerías y el ciclo de vida de todos estos componentes se entretrejaban unos a otros. Podíamos construir imágenes de máquina virtual inmutables para tener rollouts (despliegue) y rollbacks (reversible) predecibles, pero las máquinas virtuales son pesadas y poco portables.

La Manera Nueva es desplegar contenedores basados en virtualización a nivel del sistema operativo, en vez del hardware. Estos contenedores están aislados entre ellos y con el servidor anfitrión: tienen sus propios sistemas de archivos, no ven los procesos de los demás y el uso de recursos puede ser limitado. Son más fáciles de construir que una máquina virtual, y porque no están acoplados a la infraestructura y sistema de archivos del anfitrión, pueden llevarse entre nubes y distribuciones de sistema operativo.

| Ya que los contenedores son pequeños y rápidos, una aplicación puede ser empaquetada en una imagen de contenedor. Esta relación uno a uno entre aplicación e imagen nos abre un abanico de beneficios para usar contenedores. Con contenedores, podemos crear imágenes inmutables al momento de la compilación en vez del despliegue ya que las aplicaciones no necesitan componerse junto al resto del stack ni atarse al entorno de infraestructura de producción. Generar una imagen de contenedor al momento de la compilación permite tener un entorno consistente que va desde desarrollo hasta producción. De igual forma, los contenedores son más transparentes que las máquinas virtuales y eso hace que el monitoreo y la administración sean más fáciles. Esto se aprecia más cuando los ciclos de vida de los contenedores son administrados por la infraestructura en vez de un proceso supervisor escondido en el contenedor. Por último, ya que solo hay una aplicación por contenedor, administrar el despliegue de la aplicación se reduce a administrar el contenedor.

En resumen, los beneficios de usar contenedores incluyen:

- | Ágil creación y despliegue de aplicaciones: Mayor facilidad y eficiencia al crear imágenes de contenedor en vez de máquinas virtuales
- | Desarrollo, integración y despliegue continuo: Permite que la imagen de contenedor se construya y despliegue de forma frecuente y confiable, facilitando los rollbacks pues la imagen es inmutable (implementaciones y reversiones automatizadas).
- | Separación de tareas entre Dev y Ops: Puedes crear imágenes de contenedor al momento de compilar y no al desplegar, desacoplando la aplicación de la infraestructura
- | Observabilidad No solamente se presenta la información y métricas del sistema operativo, sino la salud de la aplicación y otras señales (capacidades de autocuración).
- | Consistencia entre los entornos de desarrollo, pruebas y producción: La aplicación funciona igual en un laptop y en la nube

- | Portabilidad entre nubes y distribuciones: Funciona en Ubuntu, RHEL, CoreOS, tu datacenter físico, Google Kubernetes Engine y todo lo demás
- | Administración centrada en la aplicación: Eleva el nivel de abstracción del sistema operativo y el hardware virtualizado a la aplicación que funciona en un sistema con recursos lógicos
- | Microservicios distribuidos, elásticos, liberados y débilmente acoplados: Las aplicaciones se separan en piezas pequeñas e independientes que pueden ser desplegadas y administradas de forma dinámica, y no como una aplicación monolítica que opera en una sola máquina de gran capacidad (balanceo de carga y escalado horizontal).
- | Aislamiento de recursos: Hace el rendimiento de la aplicación más predecible
- | Utilización de recursos: Permite mayor eficiencia y densidad
- | Programación automatizada.
- | Ofrece funciones listas para la empresa.
- | Infraestructura auto escalable: Puede crear una infraestructura predecible.

5. PUNTOS CLAVES

- | Docker permite la gestión de aplicaciones en contenedores que se pueden ejecutar en cualquier máquina.
- | Docker-compose nos permite ejecutar multi-contenedores de aplicaciones Dockers.
- | Kubernetes administra aplicaciones en contenedores en varios tipos de entornos físicos, virtuales y en la nube.

